

Defense Information Infrastructure (DII)

Common Operating Environment (COE)

Version 3.0/B

Application Programmer Interface (API) Reference Guide

(Digital UNIX and AIX)

February 19, 1997

Prepared for:

Defense Information Systems Agency

Prepared by:

**Inter-National Research Institute (INRI)
12200 Sunrise Valley Drive, Suite 300
Reston, Virginia 20191**

Table of Contents

Preface.....	1
1. Introduction.....	3
1.1 Overview	3
1.2 Additional Sources of Information	4
2. Calling the DII COE Tools Using the API Toolkit	5
2.1 COEAskUser.....	7
2.2 COEFindSeg.....	9
2.3 COEInstError	13
2.4 COEMsg	15
2.5 COEPrompt.....	17
2.6 COEPromptPasswd	19
2.7 COEUpdateHome.....	22
3. Interfacing with the DII COE Printer API Toolkit.....	25
3.1 Printer APIs.....	27
3.1.1 close_printer	27
3.1.2 get_printer_descriptions	29
3.1.3 get_printer_name	31
3.1.4 get_printer_type	33
3.1.5 open_printer.....	35
3.1.6 page_break.....	38
3.1.7 VDirectPrintFile	41
3.1.8 VDirectPrintMsg.....	43
3.1.9 VPrintFile	45
3.1.10 VPrintMsg	47
3.1.11 write_printer	49
3.1.12 write_printer_array.....	52
3.2 Printer API Tools.....	55
3.2.1 EM_get_current_printer_desc	55
3.2.2 EM_get_current_printer_name.....	56
3.2.3 EM_get_current_printer_type	57
Appendix A - Notes	59

This page intentionally left blank.

Preface

The following conventions have been used in this document:

[HELVETICA FONT]	Used to indicate keys to be pressed. For example, press [RETURN].
Courier Font	Used to indicate entries to be typed at the keyboard, UNIX commands, titles of windows and dialog boxes, file and directory names, and screen text. For example, execute the following command: <code>tar xvf /dev/rmt/3mn</code>
"Quotation Marks"	Used to indicate prompts and messages that appear on the screen.
Italics	Used for emphasis.

This page intentionally left blank.

1. Introduction

1.1 Overview

This document provides information and guidance needed for using Defense Information Infrastructure (DII) Common Operating Environment (COE) Version 3.0.0.3 public Application Programmer Interfaces (APIs) for the Digital UNIX 4.0 Operating System and the AIX 4.1.4 Operating System. This document contains manual pages for all of the public APIs for the DII COE toolkit for Digital UNIX 4.0 and AIX 4.1.4.

This guide is divided into the following three sections and one appendix:

Section	Page																
Introduction Provides an overview of the <i>DII COE API Reference Guide</i> and lists documents that can be referenced for further information about the DII COE APIs.	3																
Calling the DII COE Tools Using the API Toolkit Includes manual pages for the following DII COE tools: <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">COEAskUser</td> <td style="width: 50%;">COEPrompt</td> </tr> <tr> <td>COEFindSeg</td> <td>COEPromptPasswd</td> </tr> <tr> <td>COEInstError</td> <td>COEUpdateHome</td> </tr> <tr> <td>COEMsg</td> <td></td> </tr> </table>	COEAskUser	COEPrompt	COEFindSeg	COEPromptPasswd	COEInstError	COEUpdateHome	COEMsg		5								
COEAskUser	COEPrompt																
COEFindSeg	COEPromptPasswd																
COEInstError	COEUpdateHome																
COEMsg																	
Interfacing with the DII COE Printer Using the API Toolkit Includes manual pages for the following DII COE printer and printer application tools: <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">close_printer</td> <td style="width: 50%;">VprintFile</td> </tr> <tr> <td>get_printer_descriptions</td> <td>VPrintMsg</td> </tr> <tr> <td>get_printer_name</td> <td>write_printer</td> </tr> <tr> <td>get_printer_type</td> <td>write_printer_array</td> </tr> <tr> <td>open_printer</td> <td>EM_get_current_printer_desc</td> </tr> <tr> <td>page_break</td> <td>EM_get_current_printer_name</td> </tr> <tr> <td>VdirectPrintFile</td> <td>EM_get_current_printer_type</td> </tr> <tr> <td>VdirectPrintMsg</td> <td></td> </tr> </table>	close_printer	VprintFile	get_printer_descriptions	VPrintMsg	get_printer_name	write_printer	get_printer_type	write_printer_array	open_printer	EM_get_current_printer_desc	page_break	EM_get_current_printer_name	VdirectPrintFile	EM_get_current_printer_type	VdirectPrintMsg		25
close_printer	VprintFile																
get_printer_descriptions	VPrintMsg																
get_printer_name	write_printer																
get_printer_type	write_printer_array																
open_printer	EM_get_current_printer_desc																
page_break	EM_get_current_printer_name																
VdirectPrintFile	EM_get_current_printer_type																
VdirectPrintMsg																	
Notes Shows lines that are missing in the <code>DII_DEV/include/Printer/PrintAPI.h</code> file and must be added for the printer APIs and printer examples to work correctly.	59																

The DII COE toolkit APIs and printer toolkit APIs are located in separate sections. Each manual page includes a synopsis, parameters, a description, return values, notes, a reference to related functions, and an example.

Descriptions assume familiarity with the C programming language and with the DII COE development environment.

1.2 Additional Sources of Information

Reference the following documents for more information about the APIs for the DII COE toolkit:

- © *Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification* Version 2.0, DII COE I&RTS:Rev 2.0, Inter-National Research Institute, October 23, 1995
- © *Defense Information Infrastructure (DII) Common Operating Environment (COE) Version 3.0/B Programming Guide (Digital UNIX and AIX)*, DII.30B.DECIBM.PG-1, Inter-National Research Institute, February 19, 1997.

2. Calling the DII COE Tools Using the API Toolkit

The API toolkit provides developers with an interface to the DII COE runtime tools. A program can link with a public API to display and retrieve segment information. This section defines the interface for each call, including return values and necessary parameters. This section provides information about using the following APIs:

- COEAskUser
- COEFindSeg
- COEInstError
- COEMsg
- COEPrompt
- COEPromptPasswd
- COEUpdateHome.

NOTE: Reference Appendix A, *Notes*, for more information about using COEPromptPasswd.

The format of each manual page is as follows:

NAME

Function NameCProvides a brief description of the function.

SYNOPSIS

Presents the calling syntax for the routine, including the declarations of the arguments and the return type. For example:

```
returntype XFunctionName (type1 *arg1, type2 *arg2, type3 *arg3);
```

PARAMETERS

Describes each of the parameters used by the function.

DESCRIPTION

Describes what the function does and what events or side effects it causes.

RETURNS

Describes what the function returns.

NOTE

Provides any notes about the function.

SEE ALSO

Provides a reference to related functions.

EXAMPLE

Provides an example of how to use the function.

2.1 COEAskUser

NAME

COEAskUserCCOEAskUser() displays a window with a question and two answer buttons.

SYNOPSIS

```
#include <DIITools.h>

int COEAskUser
(
    char *question,
    char *b1_label,
    char *b2_label
);
```

PARAMETERS

char *question

 question - Null terminated string. The question to present to the user.

char *b1_label

 b1_label - Null terminated string. The equivalent of the `Yes` button; selection causes the function to return a 1 (TRUE).

char *b2_label

 b2_label - Null terminated string. The equivalent of the `No` button; selection causes the function to return a 0 (FALSE).

DESCRIPTION

The COEAskUser library function creates an interface that prompts the user for a question and gives a choice of `Yes` or `No` buttons to select, where `Yes` and `No` are the default labels assigned. The message question and the answer button labels can be assigned by the user. Null values for `b1_label` and `b2_label` will display default button labels.

RETURNS

COEFAILURE

Failure - The interface could not be displayed. On returning COEFAILURE, COEerrno is set to:

COEERR_NO_DISPLAY - Window could not be displayed.

COEERR_NO_MESSAGE - Question string not passed in.

TRUE

True - The equivalent `Yes` button selected by the user.

FALSE

False - The equivalent `No` button selected by the user.

NOTE

To support multi-line text, place `a \` followed by a [RETURN] in the appropriate location(s).

SEE ALSO

COEMsg, COEPrompt, COEPromptPasswd, and COEUpdateHome.

EXAMPLE: COEAskUser

To build this routine, use the following command (substitute your location for Motif libraries and

includes and the DII_DEV directory):

/*

To build this routine use the following command(substitute your location for the DII_DEV directory, Motif libraries and includes) :

HP:

```
cc -Aa -o COEAskUser_example COEAskUser_example.c -I/h/DII_DEV/include -  
I/usr/include/Motif1.2 -I/usr/include/X11R5 -L/h/DII_DEV/libs -lCOETools -lCOE  
-lPrintClient -L/usr/lib/X11R5 -L/usr/lib/Motif1.2 -lXm -lXt -lX11
```

SOLARIS:

```
cc -o COEAskUser_example COEAskUser_example.c -I/h/DII_DEV/include -  
I/usr/include/Motif1.2 -I/usr/openwin/include -L/h/DII_DEV/libs -lCOETools -  
lCOE -lPrintClient -L/usr/openwin/lib -L/usr/lib/Motif1.2 -lXm -lXt -lX11 -  
lgen
```

NT:

In your compile environment, make sure your include file path includes DII_DEV/include and your library path includes DII_DEV/libs. Link COEUserPrompts.lib during compilation. (The COEUserPrompts.dll will be required during execution.)

*/

```
#include <stdio.h>  
#include <DIITools.h>
```

```
*****  
/* COEAskUser_example */  
*****  
int main(int argc, char *argv[]){  
    char b1_lab[] = "MY_YES";  
    char b2_lab[] = "MY_NO";  
    char message[]="This is my test Message";  
    int ret_val;  
  
    /* Call DII/COE Library Function */  
    ret_val = COEAskUser(message, b1_lab, b2_lab);  
    exit(ret_val);  
}
```

2.2 COEFindSeg

NAME

COEFindSegCCOEFindSeg() returns information about a requested segment.

SYNOPSIS

```
#include <DIITools.h>

int COEFindSeg
(
    int use_installed,
    char *in_segdir,
    char *in_segname,
    char *prefix,
    char *seg_type,
    char *seg_attrib,
    char *actual_dir
);
```

PARAMETERS

```
int use_installed
    use_installed - Installed segments only - 1, any segments - 0.
char *in_segdir
    in_segdir - Null terminated string. The full directory path leading up to and including
    segdir.
char *in_segname
    in_segname - Null terminated string. The segment name of the segment residing in
    segdir.
char *prefix
    prefix - Segment prefix returned.
char *seg_type
    seg_type - Segment type returned.
char *seg_attrib
    seg_attrib - Segment attribute returned if available.
char *actual_dir
    actual_dir - Segments actual directory returned.
```

DESCRIPTION

The COEFindSeg library function searches a given segment directory for a given segment name to see if it is on disk and can be read. If no directory is specified, COEFindSeg searches for installed segments by default. When the segment is found, COEFindSeg returns the segment directory, name, prefix, type, and attribute (if the segment has an attribute).

RETURNS

COESUCCESS

Success - The function was successful in finding the requested segment.

COEFAILURE

Failure - The function was not successful in finding the requested segment. On returning COEFAILURE, COEerrno is set to:

COEERR_SEG_NOT_FND - Requested segment not found.

COEERR_NULL_PARAMS - Both segment name and directory parameters were NULL or empty.

COEERR_FOUND_NOT_INSTALLED - Requested segment was not installed as requested.

NOTE

None.

SEE ALSO

COEAskUser, COEInstError, COEMsg, and COEUpdateHome.

EXAMPLE: COEFindSeg

To build this routine, use the following command (substitute your location for Motif libraries and includes and the DII_DEV directory):

```
/*
To build this routine use the following command(substitute your location for
the DII_DEV directory, Motif libraries and includes) :

HP:
cc -Aa -o COEFindSeg_example COEFindSeg_example.c -I/h/DII_DEV/include -
I/usr/include/Motif1.2 -I/usr/include/X11R5 -L/h/DII_DEV/libs -lCOETools -lCOE
-lPrintClient -L/usr/lib/X11R5 -L/usr/lib/Motif1.2 -lXm -lXt -lX11

SOLARIS:
cc -o COEFindSeg_example COEFindSeg_example.c -I/h/DII_DEV/include -
I/usr/include/Motif1.2 -I/usr/openwin/include -L/h/DII_DEV/libs -lCOETools -
lCOE -lPrintClient -L/usr/openwin/lib -L/usr/lib/Motif1.2 -lXm -lXt -lX11 -
lgen

NT:
In your compile environment, make sure your include file path includes
DII_DEV/include and your library path includes DII_DEV/libs. Link COECom.lib,
COESeg.lib and COETools.lib during compilation.

*/
#include <stdio.h>
#include <DIITools.h>

/***********************/
/* COEFindSeg_example */
/***********************/
int main(int argc, char *argv[])
{
    char segdir[] = "";
    char segname[] = "X Windows";
    int ret_val;
    char out_prefix[8];
    char out_segtypes[81];
    char out_segattr[81];
    char out_actualldir[257];

    /* Call DII/COE Library Function */
    ret_val = COEFindSeg(1,segdir,
    segname,out_prefix,out_segtypes,out_segattr,out_actualldir ); if( ret_val ==
    COESUCCESS )
    {
        printf("Found segment '%s' prefix '%s' type '%s' at
        '%s'\n",segname,out_prefix,out_segtypes,out_actualldir);
    }
    else
    {
        printf("Segment '%s' not found \n", segname);
    }
}
```

```
    exit(ret_val);  
}
```

2.3 COEInstError

NAME

COEInstErrorCCOEInstError() displays a window with the error message.

SYNOPSIS

```
#include <DIITools.h>

int COEInstError
(
    char *message
);
```

PARAMETERS

char *message

message - Null terminated string. Error message to display.

DESCRIPTION

The COEInstError library function creates an interface that displays an installation error message. The message can be assigned by the caller of the function. COEFAILURE is always returned. It is up to the calling application to perform appropriate cleanup and return a failed exit status to its parent program (if applicable).

RETURNS

COEFAILURE

Failure - Always returned to signal the calling process that an error has occurred. On returning COEFAILURE, COEerrno is set to:

COEERR_NO_DISPLAY - Window could not be displayed.

COEERR_NO_MESSAGE - Message string not passed in.

NOTE

To support multi-line text, place a \ followed by a [RETURN] in the appropriate location(s).

SEE ALSO

COEMsg, COEPrompt, COEPromptPasswd, and COEUpdateHome.

EXAMPLE: COEInstError

To build this routine, use the following command (substitute your location for Motif libraries and includes and the DII_DEV directory):

```
/*
To build this routine use the following command(substitute your location for
the DII_DEV directory, Motif libraries and includes) :

HP:
cc -Aa -o COEInstError_example COEInstError_example.c -I/h/DII_DEV/include -
I/usr/include/Motif1.2 -I/usr/include/X11R5 -L/h/DII_DEV/libs -lCOETools -lCOE
-lPrintClient -L/usr/lib/X11R5 -L/usr/lib/Motif1.2 -lXm -lXt -lX11

SOLARIS:
cc -o COEInstError_example COEInstError_example.c -I/h/DII_DEV/include -
I/usr/include/Motif1.2 -I/usr/openwin/include -L/h/DII_DEV/libs -lCOETools -
lCOE -lPrintClient -L/usr/openwin/lib -L/usr/lib/Motif1.2 -lXm -lXt -lX11 -
lgen

NT:
In your compile environment, make sure your include file path includes
DII_DEV/include and your library path includes DII_DEV/libs. Link
COEUserPrompts.lib during compilation. (The COEUserPrompts.dll will be
required during execution.)
*/
#include <stdio.h>
#include <DIITools.h>

***** */
/* COEInstError_example      */
***** */
int main(int argc, char *argv[])
{
    char      message[]="Cannot Access ABC Server";
    int       ret_val;

    /* Call DII/COE Library Function */
    ret_val = COEInstError(message);

    exit(ret_val);
}
```

2.4 COEMsg

NAME

COEMsgCCOEMsg() displays a window with a message of choice.

SYNOPSIS

```
#include <DIITools.h>

int COEMsg
(
    char *message
);
```

PARAMETERS

char *message

message - Null terminated string. Message to display to the user.

DESCRIPTION

The COEMsg library function creates an interface that displays a message. The message can be assigned by the caller of the function, which makes it versatile enough to be used as a message for any program.

RETURNS

COESUCCESS

Success - The function was successful in displaying the message.

COEFAILURE

Failure - The function was not successful in displaying the message. On returning

COEFAILURE, COEerrno is set to:

COEERR_NO_DISPLAY - Window could not be displayed.

COEERR_NO_MESSAGE - Question string not passed in.

NOTE

To support multi-line text, place a \ followed by a [RETURN] in the appropriate location(s).

SEE ALSO

COEAskUser, COEInstError, COEPrompt, COEPromptPasswd, and COEUpdateHome.

EXAMPLE: COEMsg

To build this routine, use the following command (substitute your location for Motif libraries and includes and the DII_DEV directory):

```
/*
To build this routine use the following command(substitute your location for
the DII_DEV directory, Motif libraries and includes) :

HP:
cc -Aa -o COEMsg_example COEMsg_example.c -I/h/DII_DEV/include -
I/usr/include/Motif1.2 -I/usr/include/X11R5 -L/h/DII_DEV/libs -lCOETools -lCOE
-lPrintClient -L/usr/lib/X11R5 -L/usr/lib/Motif1.2 -lXm -lXt -lX11

SOLARIS:
cc -o COEMsg_example COEMsg_example.c -I/h/DII_DEV/include -
I/usr/include/Motif1.2 -I/usr/openwin/include -L/h/DII_DEV/libs -lCOETools -
lCOE -lPrintClient -L/usr/openwin/lib -L/usr/lib/Motif1.2 -lXm -lXt -lX11 -
lgen

NT:
In your compile environment, make sure your include file path includes
DII_DEV/include and your library path includes DII_DEV/libs. Link
COEUserPrompts.lib during compilation. (The COEUserPrompts.dll will be
required during execution.)
*/
#include <stdio.h>
#include <DIITools.h>

/***********************/
/* COEMsg_example      */
/***********************/
int main(int argc, char *argv[])
{
    char      message[ ]="XYZ Has Been Updated";
    int       ret_val;

    /* Call DII/COE Library Function */
    ret_val = COEMsg(message);

    exit(ret_val);
}
```

2.5 COEPrompt

NAME

COEPromptCCOEPrompt() displays a window with a user-assigned prompt and an editable text area for user input.

SYNOPSIS

```
#include <DIITools.h>

int COEPrompt
(
    char *prompt,
    int max_len,
    char *text_return
);
```

PARAMETERS

```
char *prompt
    prompt - Null terminated string. Prompt text to display.
int max_len
    max_len - Maximum length of text accepted in the editable text field.
char *text_return
    text_return - Null terminated string. Text returned must be allocated to max_len +
    1.
```

DESCRIPTION

The COEPrompt library function creates an interface that displays a prompt and a text area to accept user input. The prompt can be assigned by the caller of the function. The user input will be sent to `stdout`.

RETURNS

COESUCCESS

Success - The function was successful in displaying the prompt and the text area.

COEFAILURE

Failure - The function was not successful in displaying the prompt and the text area. On returning COEFAILURE, COEerrno is set to:

COEERR_NO_DISPLAY - Window could not be displayed.

COEERR_NO_MESSAGE - Prompt string not passed in.

NOTE

To support multi-line text, place a \ followed by a [RETURN] in the appropriate location(s).

SEE ALSO

COEAskUser, COEInstError, COEMsg, COEPromptPasswd, COEUpdateHome.

EXAMPLE: COEPrompt

To build this routine, use the following command (substitute your location for Motif libraries and includes and the DII_DEV directory):

```
/*
To build this routine use the following command(substitute your location for
the DII_DEV directory, Motif libraries and includes) :

HP:
cc -Aa -o COEPrompt_example COEPrompt_example.c -I/h/DII_DEV/include -
I/usr/include/Motif1.2 -I/usr/include/X11R5 -L/h/DII_DEV/libs -lCOETools -lCOE
-lPrintClient -L/usr/lib/X11R5 -L/usr/lib/Motif1.2 -lXm -lXt -lX11

SOLARIS:
cc -o COEPrompt_example COEPrompt_example.c -I/h/DII_DEV/include -
I/usr/include/Motif1.2 -I/usr/openwin/include -L/h/DII_DEV/libs -lCOETools -
lCOE -lPrintClient -L/usr/openwin/lib -L/usr/lib/Motif1.2 -lXm -lXt -lX11 -
lgen

NT:
In your compile environment, make sure your include file path includes
DII_DEV/include and your library path includes DII_DEV/libs. Link
COEUserPrompts.lib during compilation. (The COEUserPrompts.dll will be
required during execution.)
*/
#include <stdio.h>
#include <DIIITools.h>

*****
/* COEPrompt_example */
*****
int main(int argc, char *argv[])
{
    char      message[]="Please Enter What Server To Access : ";
    int       input_text_len = 15;
    int       ret_val;
    char      ret_text[16];

    /* Call DII/COE Library Function */
    ret_val = COEPrompt(message, input_text_len, ret_text);

    exit(ret_val);
}
```

2.6 COEPromptPasswd

NAME

COEPromptPasswd CCOEPromptPasswd() displays a window with an optional Password prompt message and an optional Verify prompt.

SYNOPSIS

```
#include <DIITools.h>

int COEPromptPasswd
(
    int max_text_len,
    int user_wants_verify_prompt,
    char *prompt,
    char *passwd_return
);
```

PARAMETERS

```
int max_text_len
    max_text_len - Maximum characters the user may enter for the password.
int user_wants_verify_prompt
    user_wants_verify_prompt - Flag whether to display a verify prompt.
char *prompt
    prompt - Null terminated string. Optional prompt to display text.
char *passwd_return
    passwd_return - Password returned.
```

DESCRIPTION

The COEPromptPasswd library function creates an interface that displays an optional Password prompt message and an optional Verify prompt. The optional Verify prompt can either be (1) displayed by passing in TRUE or (2) not displayed by passing in FALSE through the third parameter. The password entered will be sent to stdout.

RETURNS

COESUCCESS

Success - The function was successful. (This function is always successful if the verify prompt is not displayed.)

COEFAILURE

Failure - The interface cannot be displayed, or the Password and Verify text are not the same. On returning COEFAILURE, COEerrno is set to:

COEERR_NO_DISPLAY - Window could not be displayed.

COEERR_EMPTY_FIELD - No Password text or verify text returned.

COEERR_NO_MATCH - Password and Verify string do not match.

NOTE

The password has a default maximum length of 40 characters when max_text_len is less than or equal to zero. The maximum password length can be increased or decreased by the caller of the function via the max_text_len field. If the input maximum length is exceeded, the input will be

truncated.

To support multi-line text, place `a \` followed by a [RETURN] in the appropriate location(s).

SEE ALSO

`COEAskUser`, `COEInstError`, `COEMsg`, `COEPrompt`, and `COEUpdateHome`.

EXAMPLE: COEPromptPasswd

To build this routine, use the following command (substitute your location for Motif libraries and includes and the DII_DEV directory):

```
/*
To build this routine use the following command(substitute your location for
the DII_DEV directory, Motif libraries and includes) :

HP:
cc -Aa -o COEPromptPasswd_example COEPromptPasswd_example.c -
I/h/DII_DEV/include -I/usr/include/Motif1.2 -I/usr/include/X11R5 -
L/h/DII_DEV/libs -lCOETools -lCOE -lPrintClient -L/usr/lib/X11R5 -
L/usr/lib/Motif1.2 -lXm -lXt -lX11

SOLARIS:
cc -o COEPromptPasswd_example COEPromptPasswd_example.c -I/h/DII_DEV/include -
I/usr/include/Motif1.2 -I/usr/openwin/include -L/h/DII_DEV/libs -lCOETools -
lCOE -lPrintClient -L/usr/openwin/lib -L/usr/lib/Motif1.2 -lXm -lXt -lX11 -
lgen

NT:
In your compile environment, make sure your include file path includes
DII_DEV/include and your library path includes DII_DEV/libs. Link
COEUserPrompts.lib during compilation. (The COEUserPrompts.dll will be
required during execution.)
*/
#include <stdio.h>
#include <DIITools.h>

***** */
/* COEPromptPasswd_example */
*****/
int main(int argc, char *argv[])
{
    int      input_text_len = 10;
    int      ret_val;
    char     ret_passwd[11];

    /* Call DII/COE Library Function */
    ret_val = COEPromptPasswd(input_text_len, 1,
    "Please enter passwd for MyUser", ret_passwd); if (ret_val == COESUCCESS) {
    printf("Password Is Correct\n");
    }
    else {
    printf("Password Is Incorrect\n");
    }

    exit(ret_val);
}
```

2.7 COEUpdateHome

NAME

COEUpdateHome CCOEUpdateHome() updates the home environment variable within a script file to point to where the segment was actually installed.

SYNOPSIS

```
#include <DIITools.h>

int COEUpdateHome
(
    char *script_name,
    char *env_var
);
```

PARAMETERS

```
char *script_name
    script_name - Null terminated string. Name of the script file to be updated.
char *env_var
    env_var - Null terminated string. Environment variable to be updated.
```

DESCRIPTION

The COEUpdateHome library function modifies the given environment variable in the given script by setting it to the new home directory of a segment. COEUpdateHome is intended to be invoked from within a segment's PostInstall script to adjust to any change in the segment's home directory that may occur during installation.

RETURNS

COESUCCESS

Success - The environment variable has been successfully modified to the new home directory.

COEFAILURE

Failure - Failed to modify the environment variable because the script file does not exist, or the environment variable does not exist, or unable to determine the correct home directory.

NOTE

The new home directory is determined by checking the current directory where COEUpdateHome is invoked. If it is invoked from within the PostInstall script, the home directory will be the current directory without the SegDescrip subdirectory. If it is invoked from the command line, the home directory will be the current directory.

SEE ALSO

COEAskUser, COEFindSeg, COEInstError, COEMsg, COEPrompt, and COEPromptPasswd.

EXAMPLE: COEUpdateHome

To build this routine, use the following command (substitute your location for Motif libraries and includes and the DII_DEV directory):

```
/*
```

```
To build this routine use the following command(substitute your location for  
the DII_DEV directory, Motif libraries and includes) :
```

```
HP:
```

```
cc -Aa -o COEUpdateHome_example COEUpdateHome_example.c -I/h/DII_DEV/include -  
I/usr/include/Motif1.2 -I/usr/include/X11R5 -L/h/DII_DEV/libs -lCOETools -lCOE  
-L/usr/lib/X11R5 -L/usr/lib/Motif1.2 -lXm -lXt -lX11
```

```
SOLARIS:
```

```
cc -o COEUpdateHome_example COEUpdateHome_example.c -I/h/DII_DEV/include -  
I/usr/include/Motif1.2 -I/usr/openwin/include -L/h/DII_DEV/libs -lCOETools -  
lCOE -lPrintClient -L/usr/openwin/lib -L/usr/lib/Motif1.2 -lXm -lXt -lX11 -  
lgen */  
#include <stdio.h>  
#include <DIITools.h>
```

```
*****
```

```
/* COEUpdateHome_example      */
```

```
*****
```

```
int main(int argc, char *argv[])
{
    char script_name[] = "Scripts/.cshrc";
    char env_var[] = "MY_HOME";
    int ret_val;

    /* Call DII/COE Library Function */
    ret_val = COEUpdateHome(script_name, env_var);

    exit(ret_val);
}
```

This page intentionally left blank.

3. Interfacing with the DII COE Printer API Toolkit

The DII COE printer API toolkit provides developers with an interface to the DII COE printers. A program can link with a public API to print information. This section defines the interface needed for each call, including return values and necessary parameters.

Each DII COE printer API belongs in one of two groups. The following lower level printer APIs belong in the first group and are described in Section 3.1, *Printer APIs*: close_printer, get_printer_descriptions, get_printer_name, get_printer_type, open_printer, page_break, write_printer, and write_printer_array. The following higher level printer APIs belong in the second group and are also described in Section 3.1, *Printer APIs*: VDirectPrintFile, VDirectPrintMsg, VPrintFile, and VPrintMsg.

It is recommended that you not use APIs from both groups within the same application when using the DII COE printer APIs. For example, if you use the open_printer API, which belongs in the first group, you should not then use the VDirectPrintFile API, which belongs in the second group.

The following printer API tools provide current printer information and are described in Section 3.2, *Printer API Tools*: EM_get_current_printer_desc, EM_get_current_printer_name, and EM_get_current_printer_type. These executable programs provide the same functionality as the C functions of the same name.

Detailed information about using the DII COE printer APIs and printer API tools is provided in the following subsections.

NOTE: Please reference Appendix A, *Notes*, for more information about using the COE printer API toolkit.

The format of each manual page is as follows:

NAME

Function NameCProvides a brief description of the function.

SYNOPSIS

Presents the calling syntax for the routine, including the declarations of the arguments and return type. For example:

```
returntype XFunctionName (type1 *arg1, type2 *arg2, type3 *arg3);
```

PARAMETERS

Describes each of the parameters used by the function.

DESCRIPTION

Describes what the function does and what events or side effects it causes.

RETURNS

Describes what the function returns.

NOTE

Provides any notes about the function.

SEE ALSO

Refers to related functions.

EXAMPLE

Provides an example of how some of the most common functions are used.

3.1 Printer APIs

3.1.1 close_printer

NAME

close_printerCclose_printer finishes a print job and sends it to the printer.

SYNOPSIS

```
#include <Printer/PrintAPI.h>
int close_printer
(
    char **file_name,
    FILE **file_pointer
);
```

PARAMETERS

char **file_name

Pointer to a null terminated character string returned from open_printer containing the address of the name of a temporary file to be printed.

FILE **file_pointer

Address of the file handle of a temporary file to be printed, which is returned from open_printer.

DESCRIPTION

The close_printer function is used to conclude a single print job and send the data to the printer. If close_printer is not called, the print job will not be printed. A single print job referenced by file_pointer is closed and queued for printing. The temporary file created by open_printer is deleted upon completion of the print job.

RETURNS

PRT_SUCCESS on success; PRT_FAILURE if an error occurred.

NOTE

The area pointed to by file_pointer should be released using the free(3) library call when it is no longer needed.

SEE ALSO

get_printer_name, get_printer_type, get_printer_descriptions, open_printer, page_break, write_printer_array, and write_printer.

EXAMPLE: close_printer

To build this routine, use the following command (substitute your location for libraries and includes and the DII_DEV directory):

```
/*
cc -Aa -o close_printer_example close_printer_example.c -I/h/DII_DEV/include
-L/h/DII_DEV/libs -lPrintClient
*/
#include <stdio.h>
#include <stdlib.h>
#include <Printer/PrintAPI.h>

***** 
/* close_printer_example      */
***** 
int main()
{
    char *tmp_filename;          /* Holds temporary print file name */
    FILE *file_p;               /* Holds the file pointer to the temp file */

    /*
     * open_printer() opens printer with security level "Unclassified",
     * "80" columns/line, "60" lines/page,single spacing of lines, with "0" left
     * indentation of each line of text. "tmp_filename" is the complete path
     * of the temporary file created to contain the printed text, and "file_p"
     * is the file pointer to the temporary file that is being printed.
     */
    if(open_printer("Unclassified",80,60,1,0,&tmp_filename,&file_p) ==
       PRT_SUCCESS) {
        /*
         * close_printer() prints the file "tmp_filename" and then
         * closes the file.
         */
        if(close_printer(&tmp_filename,&file_p) != PRT_SUCCESS) {
            /*
             * If close_printer() fails, free malloced memory and exit
             */
            free(tmp_filename);
            exit(PRT_FAILURE);
        }
        /*
         * If all functions succeeded, then free malloced memory and exit
         */
        free(tmp_filename);
        exit(PRT_SUCCESS);
    }
    else {
        exit(PRT_FAILURE);
    }
    return PRT_SUCCESS;
}
```

3.1.2 get_printer_descriptions

NAME

get_printer_descriptionsC`get_printer_descriptions` returns a description of the current default printer.

SYNOPSIS

```
#include <Printer/PrintAPI.h>
int get_printer_descriptions(char **c_printer_description);
```

PARAMETERS

`char **c_printer_description`

Returns a pointer to a null terminated character string containing the description of the current default printer.

DESCRIPTION

The `get_printer_descriptions` function allows an application to retrieve the description of the default printer.

RETURNS

`PRT_SUCCESS` on success; `PRT_FAILURE` if an error occurred.

NOTE

The area pointed to by `c_printer_description` should be released using the `free(3)` library call when it is no longer needed.

SEE ALSO

`get_printer_name`, `get_printer_type`, `open_printer`, `page_break`, `write_printer_array`, `write_printer`, and `close_printer`.

EXAMPLE: get_printer_descriptions

To build this routine, use the following command (substitute your location for libraries and includes and the DII_DEV directory):

```
/*
cc -Aa -o get_printer_description_example
get_printer_descriptions_example.c -I/h/DII_DEV/include
-L/h/DII_DEV/libs -lPrintClient
-lCommon
*/
#include <stdlib.h>
#include <Printer/PrintAPI.h>

***** */
/*  get_printer_descriptions_example    */
***** /
int main()
{
    char *prtdesc;

    /* Call DII/COE Library Function */
    if(get_printer_descriptions(&prtdesc) == PRT_SUCCESS) {
        printf("%s\n", prtdesc);
        free(prtdesc);
        return PRT_SUCCESS;
    }
    else {
        return PRT_FAILURE;
    }
}
```

3.1.3 get_printer_name

NAME

`get_printer_name`C`get_printer_name` retrieves the name of the current default printer selected by the user.

SYNOPSIS

```
#include <Printer/PrintAPI.h>
int get_printer_name(char **c_printer_name);
```

PARAMETERS

`char **c_printer_name`;

Returns a pointer to a null terminated character string containing the name of the current default printer.

DESCRIPTION

The `get_printer_name` function allows an application to retrieve the name of the default printer.

RETURNS

`PRT_SUCCESS` on success; `PRT_FAILURE` if an error occurred.

NOTE

The area pointed to by `c_printer_name` should be released using the `free(3)` library call when it is no longer needed.

SEE ALSO

`get_printer_type`, `get_printer_descriptions`, `open_printer`, `page_break`, `write_printer_array`, `write_printer`, and `close_printer`.

EXAMPLE: get_printer_name

To build this routine, use the following command (substitute your location for libraries and includes and the DII_DEV directory):

```
/*
cc -Aa -o get_printer_name_example get_printer_name_example.c
-I/h/DII_DEV/include -L/h/DII_DEV/libs -lPrintClient
*/
#include <stdlib.h>
#include <Printer/PrintAPI.h>

*****get_printer_name_example*****
int main()
{
    char *prtname;

    /* Call DII/COE Library Function */
    if(get_printer_name(&prtname) == PRT_SUCCESS) {
        printf("%s\n", prtname);
        free(prtname);
        return PRT_SUCCESS;
    }
    else {
        return PRT_FAILURE;
    }
}
```

3.1.4 get_printer_type

NAME

`get_printer_type`C`get_printer_type` retrieves the type of current default printer.

SYNOPSIS

```
#include <Printer/PrintAPI.h>
int get_printer_type(char **c_printer_type);
```

PARAMETERS

`char **c_printer_type;`

Returns a pointer to a null terminated character string containing the type of current default printer.

DESCRIPTION

The `get_printer_type` function allows an application to retrieve the type of default printer. The type is either (1) ASCII, (2) HPGL, or (3) PostScript.

RETURNS

`PRT_SUCCESS` on success; `PRT_FAILURE` if an error occurred.

NOTE

The area pointed to by `c_printer_type` should be released using the `free(3)` library call when it is no longer needed.

SEE ALSO

`get_printer_name`, `get_printer_descriptions`, `open_printer`, `page_break`, `write_printer_array`, `write_printer`, and `close_printer`.

EXAMPLE: get_printer_type

To build this routine, use the following command (substitute your location for libraries and includes and the DII_DEV directory):

```
/*
cc -Aa -o get_printer_type_example
get_printer_type_example.c -I/h/DII_DEV/include
-L/h/DII_DEV/libs -lPrintClient
*/
#include <stdlib.h>
#include <Printer/Printer.h>
#include <Printer/PrintAPI.h>

***** */
/*  get_printer_type_example   */
***** */

int main()
{
    char *prttype;

    if(get_printer_type(&prttype) == PRT_SUCCESS) {
        printf("%s\n", prttype);
        free(prttype);
        return PRT_SUCCESS;
    }
    else {
        return PRT_FAILURE;
    }
}
```

3.1.5 open_printer

NAME

open_printerCopen_printer opens a connection to a printer.

SYNOPSIS

```
#include <Printer/PrintAPI.h>
int open_printer
(
    char *xcp_security_level,
    int xi_line_length,
    int xi_page_length,
    int xi_line_spacing,
    int xi_indent,
    char **xcp_file_name,
    FILE **xfp
);
```

PARAMETERS

char *xcp_security_level

Security level of a document, used for header and footer lines. xcp_security_level is a null terminated character string. It is the caller's responsibility to determine and document the proper security level of the system and to format the classification string.

int xi_line_length

Line length for a new print job. This is the maximum number of characters for each line of printed text, including the indentation characters. xi_line_length of 0 selects the default line length of 80.

int xi_page_length

Page length for a new print job. This is the number of lines of text to print per page.
xi_page_length of 0 selects the default page length of 60.

int xi_line_spacing

Line spacing for a new print job. This is the number of line feeds per line of text. A line spacing of 1 provides single spaced output. A line spacing of 2 provides double spaced output. xi_line_spacing of 0 selects single spacing, which is the default.

int xi_indent

Indentation width for a new print job. Number of characters to left indent each line of text.

char **xcp_file_name

Returns the complete path of the temporary print file created by open_printer to contain the printed text.

FILE **xfp

Returns a file pointer open to the temporary print file created by open_printer. This file is closed when the close_printer() API is called.

DESCRIPTION

The open_printer function establishes a print context, including the security level, line length, page length, line spacing, and indentation for the print job. These variables are used to format the job before sending it to the printer. This function returns a file pointer through its last argument, and this file pointer is used for all subsequent actions on this print job.

RETURNS

PRT_SUCCESS on success; PRT_FAILURE if an error occurred.

NOTE

The area pointed to by `file_name` is allocated dynamically using the malloc(3) library function and should be returned by the free(3) library function following the close_printer() API call.

SEE ALSO

`get_printer_name`, `get_printer_type`, `get_printer_descriptions`, `page_break`, `write_printer_array`, `write_printer`, and `close_printer`.

EXAMPLE: open_printer

To build this routine, use the following command (substitute your location for libraries and includes and the DII_DEV directory):

```
/*
cc -Aa -o open_printer_example open_printer_name_example.c
-I/h/DII_DEV/include -L/h/DII_DEV/libs -lPrintClient
*/
#include <stdio.h>
#include <stdlib.h>
#include <Printer/PrintAPI.h>

/***********************/
/* open_printer_example */
/***********************/
int main()
{
    char *tmp_filename;           /* Holds temporary print file name */
    FILE *file_p;                /* Holds the file pointer to the temp file */

    /*
     * open_printer() opens printer with security level "Unclassified",
     * "80" columns/line, "60" lines/page,single spacing of lines, with "0" left
     * indentation of each line of text. "tmp_filename" is the complete path
     * of the temporary file created to contain the printed text, and "file_p"
     * is the file pointer to the temporary file that is being printed.
     */
    if(open_printer("Unclassified",80,60,1,0,&tmp_filename,&file_p) ==
       PRT_SUCCESS) {

        /*
         * close_printer() prints the file "tmp_filename" and then
         * closes the file.
         */
        if(close_printer(&tmp_filename,&file_p) != PRT_SUCCESS) {
            /*
             * If close_printer() fails, free malloced memory and exit
             */
            free(tmp_filename);
            exit(PRT_FAILURE);
        }

        /*
         * If all functions succeeded, then free malloced memory and exit
         */
        free(tmp_filename);
        exit(PRT_SUCCESS);
    }
    else {
        exit(PRT_FAILURE);
    }
    return PRT_SUCCESS;
}
```

3.1.6 page_break

NAME

`page_break`C`page_break` inserts a page break in a print job.

SYNOPSIS

```
#include <Printer/PrintAPI.h>
int page_break(FILE **xfp);
```

PARAMETERS

`FILE **xfp`
File handle obtained from `open_printer`.

DESCRIPTION

The `page_break` function is used to indicate that subsequent data sent to a print job should begin on a new page.

RETURNS

`PRT_SUCCESS` on success; `PRT_FAILURE` if an error occurred.

NOTE

None.

SEE ALSO

`get_printer_name`, `get_printer_type`, `get_printer_descriptions`, `open_printer`, `write_printer_array`, `write_printer`, and `close_printer`.

EXAMPLE: page_break

To build this routine, use the following command (substitute your location for libraries and includes and the DII_DEV directory):

```
/*
cc -Aa -o page_break page_break_example.c
-I/h/DII_DEV/include -L/h/DII_DEV/libs -lPrintClient
*/
#include <stdio.h>
#include <stdlib.h>
#include <Printer/PrintAPI.h>

*****page_break_example*****
int main()
{
    char *tmp_filename; /* Holds temporary print file name */
    FILE *file_p;        /* Holds the file pointer to the temp file */

    /*
     * open_printer() opens printer with security level "Unclassified",
     * "80" columns/line, "60" lines/page,single spacing of lines, with "0" left
     * indentation of each line of text. "tmp_filename" is the complete path
     * of the temporary file created to contain the printed text, and "file_p"
     * is the file pointer to the temporary file that is being printed.
     */
    if(open_printer("Unclassified",80,60,1,0,&tmp_filename,&file_p) ==
       PRT_SUCCESS) {
        /*
         * Add a page break to the file pointed to by "file_p".
         */
        if(page_break(&file_p) != PRT_SUCCESS) {
            /*
             * If page_break() fails, free malloced memory and return
             * PRT_FAILURE.
             */
            free(tmp_filename);
            exit(PRT_FAILURE);
        }
        /*
         * close_printer() prints the file "tmp_filename" and then
         * closes the file.
         */
        if(close_printer(&tmp_filename,&file_p) != PRT_SUCCESS) {
            /*
             * If close_printer() fails, free malloced memory and return
             * PRT_FAILURE.
             */
            free(tmp_filename);
            exit(PRT_FAILURE);
        }
        /*
         * If all functions succeeded, then free malloced memory and return
         */
    }
}
```

```
* PRT_SUCCESS.  
*/  
free(tmp_filename);  
exit (PRT_SUCCESS);  
}  
else {  
    exit (PRT_FAILURE);  
}  
}
```

3.1.7 VDirectPrintFile

NAME

VDirectPrintFileCVDirectPrintFile sends a file to the specified printer.

SYNOPSIS

```
#include <Printer/PrintAPI.h>

int VDirectPrintFile
(
    char *filename,
    int prt_rec
);
```

PARAMETERS

char *filename

Name of file to print. Null terminated character string.

int prt_rec

Printer to use (-1 for default).

DESCRIPTION

The VDirectPrintFile function prints directly to the specified printer. The printer number is specified as a return value from a previous VPrintXXX or VDirectPrintXXX function call. The printer number is -1 for the DII COE default printer.

RETURNS

Returns (-1) if canceled or if an error occurred; otherwise, returns the number of the printer used.

NOTE

No classification banner will be printed with the job.

SEE ALSO

VPrintMsg, VDirectPrintMsg, and VPrintFile.

EXAMPLE: VDirectPrintFile

To build this routine, use the following command (substitute your location for libraries and includes and the DII_DEV directory):

```
/*
cc -Aa -o VDirectPrintFile_example VDirectPrintFile_example.c
-I/h/DII_DEV/include -L/h/DII_DEV/libs -lPrintClient
*/
#include <stdio.h>
#include <Printer/Printer.h>
#include <Printer/PrintAPI.h>

***** 
/*  VDirectPrintFile_example   */
***** 
int main(int argc, char *argv[])
{
    int i;
    int err;

    for(i=1;i<argc;i++) {
        /* Call DII/COE Library Function */
        err = VDirectPrintFile(argv[i], -1);
        if (err != -1) {
            printf("file %s sent to printer\n", argv[i]);
        }
        else {
            printf("error\n");
        }
    }
}
```

3.1.8 VDirectPrintMsg

NAME

VDirectPrintMsgCVDirectPrintMsg sends an internal text buffer to the default printer.

SYNOPSIS

```
#include <Printer/PrintAPI.h>

int VDirectPrintMsg
(
    char **msg_array,
    int nlines
);
```

PARAMETERS

char **msg_array

 Array of null terminated text strings.

int nlines

 Number of lines in the array.

DESCRIPTION

The VDirectPrintMsg function prints a text message directly to the DII COE default printer.

RETURNS

Returns (-1) if canceled or if an error occurred; otherwise, returns the number of the printer used.

NOTE

No classification banner will be printed with the job.

SEE ALSO

VPrintMsg, VPrintFile, and VDirectPrintFile.

EXAMPLE: VDirectPrintMsg

To build this routine, use the following command (substitute your location for libraries and includes and the DII_DEV directory):

```
/*
cc -Aa -o VDirectPrintMsg_example VDirectPrintMsg_example.c
-I/h/DII_DEV/include -L/h/DII_DEV/libs -lPrintClient
*/
#include <stdio.h>
#include <Printer/PrintAPI.h>

/*****************/
/*  VDirectPrintMsg_example   */
/*****************/
#define NUM_OF_LINES 3

int main()
{
    int err;
    char *c_string_array[NUM_OF_LINES]; /* Used with VDirectPrintMsg() */

    /*
     * Create an array of strings to pass to VDirectPrintMsg. The last
     * entry in the array must be NULL.
     */
    c_string_array[0] = "This is line 1";
    c_string_array[1] = "This is line 2";
    c_string_array[2] = NULL;
    err = VDirectPrintMsg(c_string_array, NUM_OF_LINES);
    if (err != -1) {
        printf("sent to default printer\n");
    }
    else {
        printf("error\n");
    }
}
```

3.1.9 VPrintFile

NAME

VPrintFileCVPrintFile sends a file to an operator-selected printer.

SYNOPSIS

```
#include <Printer/PrintAPI.h>

int VPrintFile(char *filename);
```

PARAMETERS

char *filename

Null terminated string. Name of the file to be printed.

DESCRIPTION

The VPrintFile function retrieves all available printers on the network and displays them in the Print Selector window for operator selection or cancellation. Once the selection has been made, the selected item can be printed.

RETURNS

Returns (-1) if canceled or if an error occurred; otherwise, returns the number of the printer used.

NOTE

None.

SEE ALSO

VPrintMsg, VDirectPrintMsg, and VDirectPrintFile.

EXAMPLE: VPrintFile

To build this routine, use the following command (substitute your location for libraries and includes and the DII_DEV directory):

```
/*
cc -Aa -o VPrintFile_example VPrintFile_example.c
-I/h/DII_DEV/include -L/h/DII_DEV/libs -lPrintClient
*/
#include <Printer/PrintAPI.h>

***** 
/*  VPrintFile_example      */
***** 
int main(int argc, char *argv[])
{
    int i;
    int err;

    for(i=1;i<argc;i++) {
        err = VPrintFile(argv[i]);
        if (err != -1) {
            printf("file %s sent to printer\n",argv[i]);
        }
        else {
            printf("error\n");
        }
    }
}
```

3.1.10 VPrintMsg

NAME

VPrintMsg C VPrintMsg sends an internal text buffer to an operator-selected printer.

SYNOPSIS

```
#include <Printer/PrintAPI.h>

int VPrintMsg
(
    char **msg_array,
    int nlines
);
```

PARAMETERS

```
char **msg_array
        Array of null terminated text strings.
int nlines
        Number of lines in the array.
```

DESCRIPTION

The VPrintMsg function retrieves all available printers on the network and displays them in the Print Selector window for operator selection or cancellation. Once the selection has been made, the selected item can be printed.

RETURNS

Returns (-1) if canceled or if an error occurred; otherwise, returns the number of the printer used.

NOTE

None.

SEE ALSO

VDirectPrintMsg, VPrintFile, and VDirectPrintFile.

EXAMPLE: VPrintMsg

To build this routine, use the following command (substitute your location for libraries and includes and the DII_DEV directory):

```
/*
cc -Aa -o VPrintMsg_example VPrintMsg_example.c
-I/h/DII_DEV/include -L/h/DII_DEV/libs -lPrintClient
*/
#include <stdio.h>
#include <Printer/PrintAPI.h>

/*****************/
/*  VPrintMsg_example          */
/*****************/
#define NUM_OF_LINES 3

int main()
{
    int i;
    int err;
    char *c_string_array[NUM_OF_LINES]; /* Used with VDirectPrintMsg() */

    /*
     * Create an array of strings to pass to VDirectPrintMsg. The last
     * entry in the array must be NULL.
     */
    c_string_array[0] = "This is line 1";
    c_string_array[1] = "This is line 2";
    c_string_array[2] = NULL;
    printf("First line sent to printer\n");
    for(i=2;i<NUM_OF_LINES;i++) {
        /* Call DII/COE Library Function */
        err = VPrintMsg(c_string_array, NUM_OF_LINES);
        if (err != -1) {
            printf("line %d sent to default printer\n",i);
        }
        else {
            printf("error\n");
        }
    }
}
```

3.1.11 write_printer

NAME

write_printerC**write_printer** adds a string to a print job.

SYNOPSIS

```
#include <Printer/PrintAPI.h>

int write_printer
(
    char *c_string,
    FILE *fp
);
```

PARAMETERS

char **c_string
Pointer to a single null terminated character string to be printed.
FILE *fp
File handle returned from **open_printer**.

DESCRIPTION

The **write_printer** function prints a character string to the print job specified by **fp**.

RETURNS

PRT_SUCCESS on success; **PRT_FAILURE** if an error occurred.

NOTE

None.

SEE ALSO

get_printer_name, **get_printer_type**, **get_printer_descriptions**, **open_printer**, **page_break**, **write_printer_array**, and **close_printer**.

EXAMPLE: write_printer

To build this routine, use the following command (substitute your location for libraries and includes and the DII_DEV directory):

```
/*
cc -Aa -o write_printer_example write_printer_example.c
-I/h/DII_DEV/include -L/h/DII_DEV/libs -lPrintClient
*/
#include <stdio.h>
#include <stdlib.h>
#include <Printer/PrintAPI.h>

/*************
/* write_printer_example */
/************/

int main()
{
    char *tmp_filename;           /* Holds temporary print file name */
    FILE *file_p;                /* Holds the file pointer to the temp file */
    char *c_string=NULL;         /* Used with write_printer() */

    /*
     * open_printer() opens printer with security level "Unclassified",
     * "80" columns/line, "60" lines/page,single spacing of lines, with "0" left
     * indentation of each line of text. "tmp_filename" is the complete path
     * of the temporary file created to contain the printed text, and "file_p"
     * is the file pointer to the temporary file that is being printed.
     */
    if(open_printer("Unclassified",80,60,1,0,&tmp_filename,&file_p) ==
       PRT_SUCCESS) {
        /*
         * write_printer() writes a string to the file pointed to by "file_p".
         */
        c_string = "This is a line";
        if (write_printer(&c_string,file_p) != PRT_SUCCESS) {
            /*
             * If write_printer() fails, free malloced memory and return
             * PRT_FAILURE.
             */
            free(tmp_filename);
            exit(PRT_FAILURE);
        }

        /*
         * close_printer() prints the file "tmp_filename" and then
         * closes the file.
         */
        if(close_printer(&tmp_filename,&file_p) != PRT_SUCCESS) {
            /*
             * If close_printer() fails, free malloced memory and return
             * PRT_FAILURE.
             */
            free(tmp_filename);
        }
    }
}
```

```
    exit(PRT_SUCCESS);
}

/*
 * If all functions succeeded, then free malloced memory and return
 * PRT_SUCCESS.
 */
free(tmp_filename);
exit(PRT_SUCCESS);

}
else {
    exit(PRT_FAILURE);
}
}
```

3.1.12 write_printer_array

NAME

`write_printer_array` adds an array of strings to a print job.

SYNOPSIS

```
#include <Printer/PrintAPI.h>

int write_printer_array
(
    char **c_string,
    FILE *fp
);
```

PARAMETERS

`char **c_string`

Array of null terminated character strings to be printed. The last string must be null to indicate the last entry.

`FILE *fp`

File handle returned by `open_printer`.

DESCRIPTION

The `write_printer_array` function writes an array of strings, separated by line feed characters, to the print job specified by `fp`.

RETURNS

`PRT_SUCCESS` on success; `PRT_FAILURE` if an error occurred.

NOTE

None.

SEE ALSO

`get_printer_name`, `get_printer_type`, `get_printer_descriptions`, `open_printer`, `page_break`, `write_printer`, and `close_printer`.

EXAMPLE: write_printer_array

To build this routine, use the following command (substitute your location for libraries and includes and the DII_DEV directory):

```
/*
cc -Aa -o write_printer_array_example write_printer_array_example.c
-I/h/DII_DEV/include -L/h/DII_DEV/libs -lPrintClient
*/
#include <stdio.h>
#include <stdlib.h>
#include <Printer/PrintAPI.h>

/*****************/
/* write_printer_array_example */
/*****************/
int main()
{
    char *tmp_filename;           /* Holds temporary print file name */
    FILE *file_p;                /* Holds the file pointer to the temp file */
    char *c_string_array[5];      /* Used with write_printer() */

    /*
     * open_printer() opens printer with security level "Unclassified",
     * "80" columns/line, "60" lines/page,single spacing of lines, with "0" left
     * indentation of each line of text. "tmp_filename" is the complete path
     * of the temporary file created to contain the printed text, and "file_p"
     * is the file pointer to the temporary file that is being printed.
     */
    if(open_printer("Unclassified",80,60,1,0,&tmp_filename,&file_p) ==
       PRT_SUCCESS) {

        /*
         * Create an array of strings to pass to write_printer_array. The last
         * entry in the array must be NULL.
         */
        c_string_array[0] = "This is line 1";
        c_string_array[1] = "This is line 2";
        c_string_array[2] = NULL;
        if (write_printer_array(c_string_array,file_p) != PRT_SUCCESS) {
            /*
             * If write_printer_array() fails, free malloced memory and exit
             */
            free(tmp_filename);
            exit(PRT_FAILURE);
        }

        /*
         * close_printer() prints the file "tmp_filename" and then
         * closes the file.
         */
        if(close_printer(&tmp_filename,&file_p) != PRT_SUCCESS) {
            /*
             * If close_printer() fails, free malloced memory and return
             */
        }
    }
}
```

```
    * PRT_FAILURE.  
    */  
    free(tmp_filename);  
    exit(PRT_FAILURE);  
}  
  
/*  
 * If all functions succeeded, then free malloced memory and return  
 * PRT_SUCCESS.  
 */  
free(tmp_filename);  
exit(PRT_SUCCESS);  
  
}  
else {  
    exit(PRT_FAILURE);  
}  
}
```

3.2 Printer API Tools

3.2.1 EM_get_current_printer_desc

NAME

EM_get_current_printer_desc

SYNOPSIS

EM_get_current_printer_desc

PARAMETERS

None.

DESCRIPTION

This C Shell script returns the text description of the currently selected printer.

RETURNS

The currently selected printer description.

NOTE

The executable script EM_get_current_printer_desc is in the /h/COE/bin directory.

SEE ALSO

EM_get_current_printer_name and EM_get_current_printer_type.

EXAMPLE: EM_get_current_printer_desc

```
#!/bin/csh  
  
printer_desc = '/EM_get_current_printer_desc'  
  
echo $printer_desc
```

3.2.2 EM_get_current_printer_name

NAME

EM_get_current_printer_name

SYNOPSIS

EM_get_current_printer_name

PARAMETERS

None.

DESCRIPTION

This C Shell script returns the name of the currently selected printer.

RETURNS

The currently selected printer name.

NOTE

The executable script EM_get_current_printer_name is in the /h/COE/bin directory.

SEE ALSO

EM_get_current_printer_desc and EM_get_current_printer_type.

EXAMPLE: EM_get_current_printer_name

```
#!/bin/csh  
  
printer_name = `EM_get_current_printer_name`  
  
echo $printer_name
```

3.2.3 EM_get_current_printer_type

NAME

EM_get_current_printer_type

SYNOPSIS

EM_get_current_printer_type

PARAMETERS

None.

DESCRIPTION

This C Shell script returns the type of currently selected printer. Valid printer types are ASCII, HPCL, and Postscript.

RETURNS

The currently selected printer type.

NOTE

The executable script EM_get_current_printer_type is in the /h/COE/bin directory.

SEE ALSO

EM_get_current_printer_desc and EM_get_current_printer_name.

EXAMPLE: EM_get_current_printer_type

```
#!/bin/csh  
  
printer_type = `./EM_get_current_printer_type`  
  
echo $printer_type
```

This page intentionally left blank.

Appendix A - Notes

Printers

The following lines are missing in the DII_DEV/include/Printer/PrintAPI.h file:

```
#ifndef _Printer_h_
#define _Printer_h_

#define PRT_SUCCESS      0
#define PRT_FAILURE     -1

#define PRT_NO_HOST      1
#define PRT_TYPE_SETUP   2
#define PRT_QUEUE_NOT_EMPTY 3
#endif
```

These lines must be added for the printer APIs and printer examples to work correctly. Use the editor of your choice and type the above lines at the end of the file.

This page intentionally left blank.